NISTIR 5824

Interoperability Experiments with CORBA and Persistent Object Base Systems

Elizabeth Fong
Deyuan Yang

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Gaithersburg, MD 20899

NIST

NISTIR 5824

Interoperability Experiments with CORBA and Persistent Object Base Systems

Elizabeth Fong
Deyuan Yang

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Gaithersburg, MD 20899

April 1996

U.S. DEPARTMENT OF COMMERCE
Michael Kantor, Secretary

TECHNOLOGY ADMINISTRATION
Mary L. Good, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Arati Prabhakar, Director

PREFACE

The Computer Systems Laboratory (CSL) of the National Institute of Standards and Technology (NIST) is assisting the Advanced Research Projects Agency (ARPA) in a technology evaluation of persistent object base system (POB). POB systems, generally referred to as Object Database Management Systems (ODBMSs), are database systems which provide storage and retrieval of data that are not necessarily tabular (with rows and columns).

Two main activities are currently being performed:

(1) Design and develop a set of functional test suites to be exercised on selected POB software, specifically those prototype POB systems which are developed with ARPA funds. One such prototype system is the Open OODB developed by Texas Instruments. This project is reported in [FONG95].

(2) Conduct interoperation experiments with Common Object Request Broker Architecture (CORBA) as distributed communication software using selected POB systems as database servers. This report describes this activity.

Certain commercial software products and companies are identified in this report for purposes of specific illustration. Such identification does not imply recommendation or endorsement by NIST, nor does it imply that the products identified are necessarily the best ones available for the purpose.

ABSTRACT

This report describes the design and development of interoperability experiments using the Common Object Request Broker Architecture (CORBA) products. The experiments will focus on establishing practical experiences for how to do "plug and play" using CORBA products with Persistent Object Base (POB) systems such as Texas Instruments' Open OODB and commercially available object database systems (ODBMS) such as MATISSE. The experiments will also investigate methodologies for integration of new or legacy distributed applications through the CORBA middleware infrastructure.

KEYWORDS: Distributed computing; Client-server; common Object Request Broker Architecture; CORBA; Object-oriented; Object Databases; Persistent Object Bases.

ACKNOWLEDGEMENTS

The ARPA POB project was started in 1990 by Erik Mettala who initially jump-started the POB evaluation and testing effort.

The major portion of the NIST contribution to the POB project is managed under the direction of Dr. Gio Wiederhold and Dr. David Gunning also of ARPA. We would like to acknowledge them for providing us with valuable technical direction into the interoperability experiments using OMG's CORBA technology.

We are very grateful for those vendors who donated their product for this project. The company names and the products are as follows:

Digital Equipment Corporation - ACA Services
MITRE Corporation - DISCUS Package
Iona Technologies - ORBIX
ADB, Incorporated - MATISSE
Texas Instruments - Open OODB
ORACLE Corporation - ORACLE

We would like to thank Kathy Harvill of NIST who assisted in maintaining the software and the scenarios for demonstration purposes, and who produced the diagrams presented in this report.

TABLE OF CONTENTS

1. INTRODUCTION

The need for interoperation between dissimilar systems is becoming more evident as large data intensive projects are being developed. These large applications include electronic commerce, digital libraries, advanced manufacturing, environmental monitoring and health. These applications require data from many sources and integrated within appropriate models of distribution.

1.1 Purpose and Scope

This report summarizes the work conducted by the Computer Systems Laboratory (CSL) of the National Institute of Standards and Technology (NIST) in support of the Advanced Research Projects Agency (ARPA) in the investigation of the integration of Persistent Object Base (POB) systems using open architecture specifications. The overall goal of the testing and evaluation of POB systems performed under this project was to explore and evaluate various techniques for the integration of existing POB systems or legacy systems through the use of middleware software.

This project focuses on the design of different scenarios to demonstrate the interoperability of software components. The goals of the interoperability experiments are:

- to explore ways of achieving interoperability between heterogeneous systems by using middleware, such as the Common Object Request Broker Architecture (CORBA),
- to develop application systems that can "plug-and-play" with the existing, commercial software components,
- to investigate and experiment with different ways of object wrapping,
- to explore useful interfaces and architectures for potential standardization in the areas of distributed computing and methods for interoperation of reusable software components.

1.2 Interoperability Issues

The general notion of interoperability is the capability of systems to communicate with one another, to potentially request services and to exchange and use information including content, format, and semantics. However, interoperability can be achieved across different levels: across platforms, across operating systems, across applications, and across communication networks.

From the perspective of computer systems users and application developers, an interoperable computing environment should be based

on "open" systems architecture. The considerations of "openness" considerations are:

- software component openness - a component is "open" if it could be integrated with other software components in a manner that is efficient and without the need for modification to the executable components themselves.

- public interface specification - an interface specification is public if it is based on interface specifications that are easily available and published in the public domain. Non-public or proprietary interface specifications tend to prevent migration or integration of components from other vendors.

- language bindings exist for interface standard - many interface specifications will require an application programming interface so that tool builders and integrators can write programs that access the services. Some standards specify such interfaces in a language-independent manner rather than in a particular programming language. To meet goals for portability, the interface specification should typically include language bindings for the major programming languages.

The benefits of using an open systems architecture are that these systems are generally not proprietary systems and users can mix-and match components.

1.3 Models of Distributed Computing

Distributed systems are coming to be regarded as the integration of distinct software components focusing on standardized interfaces or application programming interfaces (API). Distributed systems are characterized by software components and databases that are physically distributed, have heterogeneous interconnection characteristics, and use diverse mechanisms for cooperation. Many distributed models have been created over the last decade [FONG91]. The two prominent examples are the client-server computing model and distributed database systems. However, truly distributed, powerful systems that employ all-embracing data models and languages to provide transparent accesses to separate databases and computers are still not available. Current practice still depends on single vendors to supply the database management systems and communications products. The increased use of standard-based products or languages such as SQL will alleviate the vendor dependence problem to some extent.

The emergence of object-oriented technology has brought on a new model of distributed computing. The main characteristics of object computing are:

(1) objects are computational entities in which data and procedures are bundled together internally and hidden from external access and view, and

(2) communication with objects is accomplished through messages which invoke or request specific object functions using a well-specified public interface.

In a distributed object computing environment, the application systems would be comprised of objects that have public interfaces, with their private implementations hidden from external view. Uniform service interfaces called Application Program Interfaces (APIs) would be established that would go through a "middleware" service that allows an application to locate, transparently across the network, and to interact with another application or service.

The advantage of using the middleware concept is that the applications can be logically separate from the underlying network protocols. The applications can potentially change over time or be moved to different machines. The many client programs would not need to know about the change - only the registration facility in the middleware would need to know.

1.4 The Role of the Object Request Broker

The Object Management Group (OMG) developed a reference model architecture called Object Management Architecture (OMA) [SOLE90]. The primary goals of OMA are to solve the problem of improving productivity by reusing existing software components, by providing transparency over heterogeneous networks, and by allowing specific tasks to be delegated to specific machines. The Object Request Broker is the communications middleware of the OMG architecture. It provides an infrastructure that allows objects to communicate with each other independent of the specific platforms and techniques used to implement the addressed objects.

1.5 The Outline of This Report

Section 2 of this report will provide a description of the Common Object Request Broker Architecture (CORBA) and a review of selected CORBA products. Section 3 provides an overview of POBs and a review of POB products. Section 4 describes how to run CORBA and the interoperability test scenarios to test the operations of two CORBA products. Section 5 describes the interoperability experiments with CORBA that were used to access three POB systems. Section 6 summaries the results of the interoperability experiments and provides a short conclusion.

2.0 OVERVIEW OF CORBA AND CORBA PRODUCTS

The object request broker (ORB) acting as the infrastructure, or as the back-bone within the Object Management Architecture (OMA), is now known as the Common Object Request Broker Architecture (CORBA) [OMG 91]. The purpose of the ORB is to provide the mechanisms by which objects transparently make and receive requests and responses. Thus, the ORB provides interoperability between applications on different machines in heterogeneous distributed environment, and seamlessly interconnects multiple object systems.

2.1 What is CORBA?

The basic working of an ORB consists of a client, an ORB demon, and object implementations[1] which from this point on will be referred to as servers. As shown in Figure 2.1 the client issues a request for an object implementation. The object implementation is the provider of service. The request is routed and processed via the ORB. The ORB, searches through an implementation repository, finds and activates the specific server object to be executed, and then launches the server. Invocation of a server involves specifying the object to be invoked, the operation to be performed, and parameters to be given to the operation or returned from it. The ORB demon manages the control transfer and data transfer to the server and back to the client. In the event that the ORB cannot complete the invocation, an error response or exception is provided.

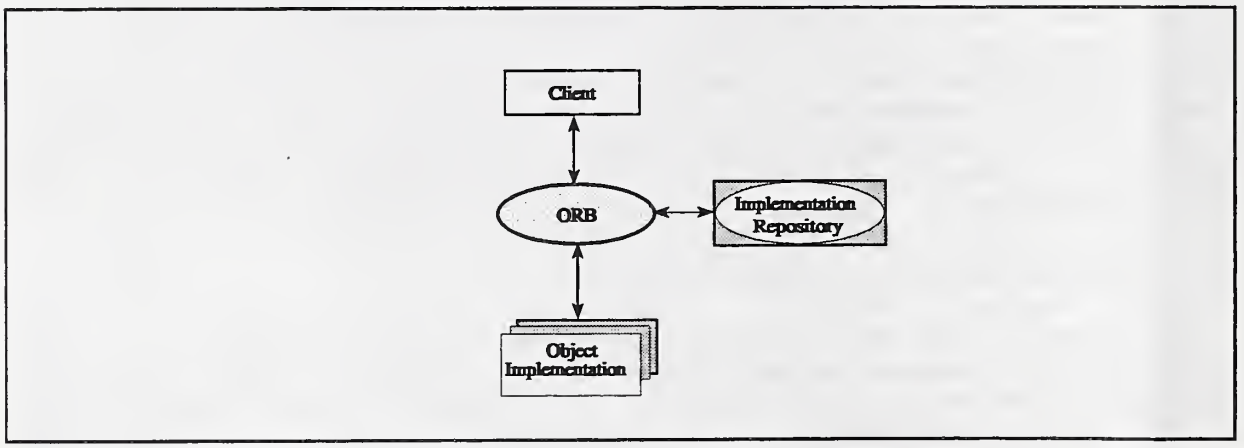

Figure 2.1 OMG ORB Model

---

[1]Object Implementation was one of the terminologies used in the earlier OMA document. Currently, other terms such as "methods" or "method server" or simply "server" are all being used.

The object implementations or servers must be defined to the ORB through well-specified interfaces using the Interface Definition Language (IDL). This language provides detailed information about the operations permitted, the arguments each such operation expects, what it returns and what happens when errors occur. The IDL is not a programming language, however, it provides the mappings to many programming languages, the first of those are C and C++.

The structure of the ORB interfaces are shown in Figure 2.2 In order for the ORB to locate the appropriate server, one must first define the server objects in IDL by specifying their interfaces. The object interface definitions are compiled by the IDL compiler resulting in the generation of the client stubs and server skeletons. Client stubs and server skeletons are terms used by the OMG to refer to fragments of code to be inserted with the application program code to be written by the application developer. The server name must also be registered in the ORB's interface repository.

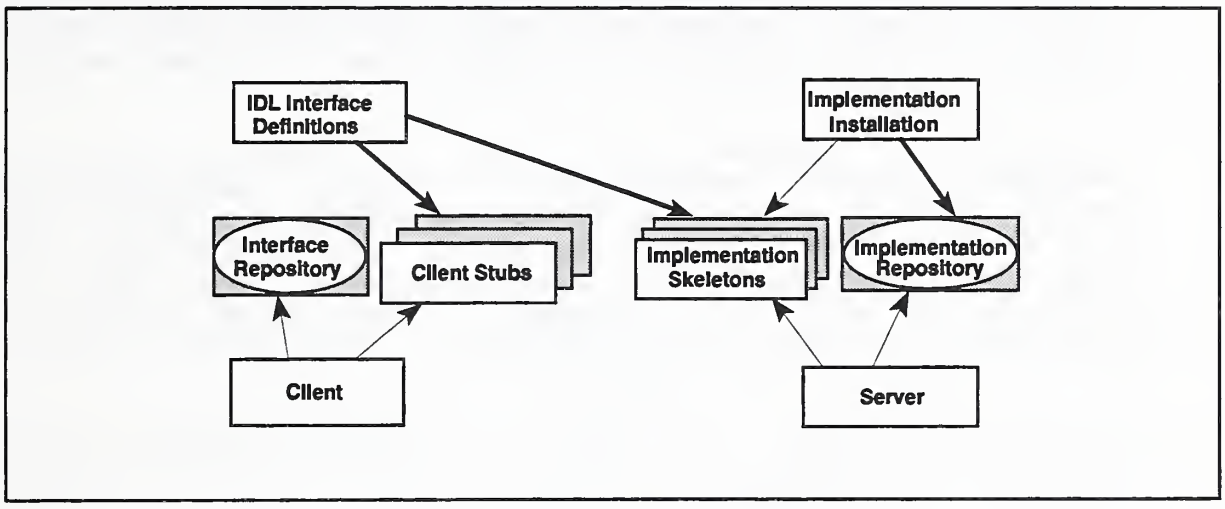

Figure 2.2 Structure of ORB Interfaces

2.2 Features of Object Request Broker

Different ORBs may make different implementation choices, thus providing a set of services to clients and implementations of objects that have different properties and qualities. The specification of CORBA is now progressed to version 2.0 which promises to support cross-ORB interoperability. The CORBA 1.2 specification calls for the following generic features to be supported to some degree:

- Name Services. Object names are used to locate the method to perform the requested operation.

- Request Dispatch. This function determines which method to invoke.

- Parameter Encoding. The ORB must have facilities to convey the local representation of parameter values in the requester's environment to equivalent representations in the recipient's environment, (i.e., from the client side to the server side).

- Delivery. Requests and results must be delivered to the proper location.

- Synchronization. Synchronization primarily deals with handling the parallelism of the object making and processing a request. Possible synchronization models include: asynchronous or one-way request (request with no response), synchronous (request where the client awaits the reply), and deferred synchronous (request sent without waiting for reply, requestor claims the response later). The CORBA specification also allows for a client to send multiple requests without waiting for the operations to finish.

- Activation. Activation is the housekeeping processing necessary before a method can be invoked. Activation and deactivation of objects is needed to obtain the object state for use when the object is accessed, and to save the state when it no longer needs to be accessed. When there are multiple objects or implementations active, rules or activation policies must be defined. Different ORB products may have different activation policies.

- Exception Handling. The ORB must be able to handle various failures in the process of object location, and to coordinate recovery housekeeping activities.

- Security Mechanism. The ORB must provide security enforcement mechanisms that support security policies. Also the ORB must have protection mechanisms assuring the integrity of data being conveyed, and assuring that the data being communicated are accessible only to authorized parties.

2.3 Third Party CORBA-compliant Products

The above are features specified in the OMG's specification for CORBA 1.2. There were several implementations of CORBA products which claimed to be OMG's CORBA 1.2 compliant [OMG94].

The CORBA products directory published in March 1994 listed 19 vendors offering CORBA products.

The following three CORBA products[2] were installed and exercised at the Computer Systems Laboratory at NIST.

- Application Control Architecture System (ACAS) developed by Digital Equipment Corporation (DEC), is one of the oldest CORBA products. DEC has already replaced ACAS with a newer product called Object Request Broker. The ACAS product evaluated at CSL was the middleware that came with the Data Integration and Synergistic Collateral Usage Study (DISCUS) package [FLEI93].

- Orbeline [POST94], developed by PostModern Computing Technologies, Inc. contains all of the features described above plus a host of other features such as fault-tolerance, failure recovery and event handling.

- Orbix, developed by IONA Technologies Ltd., is a full implementation of the OMG's CORBA. In addition, ORBIX offers support for C++ programming language binding.

---

[2]The commercial products and companies identified in this report are for purposes of specific illustration. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for this purpose.

3. OVERVIEW OF POB SYSTEMS AND POB PRODUCTS

Persistent object base systems, generally referred to as Object Database Management Systems (ODBMSs), are database systems that provide storage and retrieval of data not necessarily kept in a tabular form using rows and columns. A POB system typically integrates database capabilities with object-oriented programming language capabilities.

3.1 Types of POB Systems

There is currently no consensus on a specific set of features that can be identified as POB systems. There are many articles found today in the open literature which identify some of the existing and ongoing work in the definition of features needed to support POB systems [DABR90, OODB91, CATT94]. In general, the POB systems combine the capabilities of conventional database management systems and object-oriented programming languages such as Smalltalk.

A review of the POB products in the marketplace reveals a wide range of systems that might be called POB systems. In this report, three types of POB systems are distinguished:

- persistent programming language systems
- object database management systems with object data model
- extended relational systems

3.2 Persistent Programming Language Systems

With most programming languages, data in the program's address space is transient, i.e., it exists only as long as the program executes. To make data persist beyond the lifetime of a single program execution the programmer must include explicit instructions to save the data on stable storage, such as in a file on disk or by using a database management system as an intermediary. At a later time, when the data needs to be reused, the programmer must include explicit instructions to fetch the data from stable storage or from the database.

The design of persistent programming language systems are motivated by the requirement that any instance of any programming language data type should be allowed to persist. Some of these systems also incorporated query facilities that allowed the programmer to fetch sets of data objects from the databases, in some cases following pointer-based data relationships, in one access. Also, by allowing database operations to be bracketed in transactions, they support the controlled sharing of concurrently accessed data objects.

An example of such a persistent programming language system is Open OODB by Texas Instruments.

3.3 Object Database Management Systems

The ODBMS combines capabilities of conventional DBMS and object-oriented programming technology. These types of systems differ from relational DBMS as they generally support all of the object model characteristics, such as subclassing, type extensibility, complex objects, object identification, etc. ODBMSs also support such concepts as inheritance, mechanisms to maintain multiple versions of objects, and extensive predefined object class libraries.

These types of systems possess many of the facilities which are currently in the relational DBMS model. These facilities include the basic database engine that handles the basic DBMS operations involving defining, storing, retrieving and updating data, and also includes a query language facility, usually some form of SQL facility. Other object features supported generally consist of an object class or type definition facility, built-in application programming interfaces (APIs) for defining the object methods that form part of the class definitions, an object class library, and a set of tools. An example of ODBMS is MATISSE by ADB, Inc.

3.4 Extended Relational DBMSs

Recently, a new class of DBMS emerged which attempts to combine features from an ODBMS with a relational DBMS. These types of systems are referred to as object/relational database management system (ORDBMS) [BOWE95]. ORDBMSs are based on the idea of full relational query support, with the addition of objects (abstract data types) as additional data structure concepts that can be used to generalize relational tables and types used within them. Relational products are clearly not good at complex data, and ORDBMS is build based upon the need to support complex data. Complex data includes video, audio, compound documents, animation, arrays, geographic data, and composite objects such as a CAD drawing. An example of ORDBMS is UNISQL by UNISQL Inc.

4. CORBA TEST SCENARIOS

In this section, we describe how a typical CORBA product can be setup and run. The evaluation of the two CORBA products revealed that there are fundamental differences in terms of how a CORBA product selects its server. The server activation method of ACAS and ORBIX are described here along with several generic test scenarios.

4.1 CORBA Setup

Most CORBA 1.1 products have very similar set up procedures. The description below uses ORBIX with a C++ language binding of the IDL.

Figure 4.1 describes the ORBIX application architecture. The user first writes the IDL statements, referred to in the Figure as Appl.idl. The Appl.idl statements are compiled by the ORBIX-IDL compiler. The result of the IDL compiler consists of three distinct files generated by the ORBIX:

- The application's header file, Appl.hh
- The client stub program, ApplC.cc
- The server skeleton program, ApplS.cc

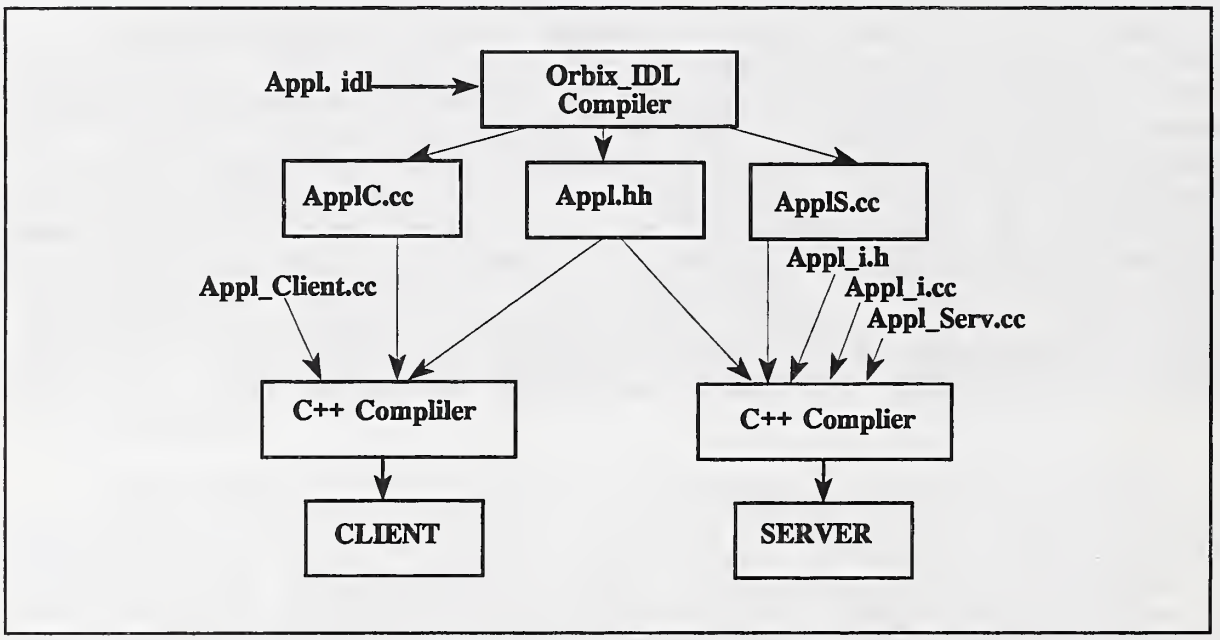

Figure 4.1 ORBIX Application Architecture

The user needs to write the following:

- The object implementation or method's class definition as Appl_i.h.
- The method code as Appl_i.cc.
- The server application code as Appl_Serv.cc
- The client application code as Appl_Client.cc

All of these programs are compiled by the C++ compiler as the client object program and as the server object program. These programs are ready to be executed when the client receives the appropriate request and invokes an object method server.

4.2 Registration of Server and Run Application

When the compiled client application code and the compiled server code are ready, the server name must be registered with the CORBA. The server name must be the interface name as defined in the IDL statements. This is accomplished by issuing the registration command of the CORBA product. The server name is now included in the interface repository of the CORBA.

To run the application, the client program is executed which will make a request for a particular server method. The needed server is activated by the CORBA based upon the product's activation policy. The server may be resident on the same machine with the client or it could be resident on a remote machine. If the server is on a remote machine, that remote machine must also have a copy of the CORBA demon. The server is launched and the result or exception message is returned to the client.

4.3 Server Activation Policies

Several experiments were conducted using two different CORBA products. The scenarios were designed to test server activation policies when the client calls a particular method server. The calls for the test were static but it was determined that the behavior would be the same if the calls were dynamically constructed.

The test scenarios include the following variables:

- single machine and multiple machines,
- server registered/not registered with the ORB product,
- server in a state of active/not active.

4.3.1 Server Activation Policy of ACAS

The activation policy of ACAS is depicted in Figure 4.2. The client first issues a request for a method server. ACAS will then check whether this method server is active. Any server must be in a state of either active or deactive. If the server is active, then it is selected and invoked. If the local server not active, ACAS will go to a remote node to check whether there is an active requested method server available. If yes, then the remote server is selected and invoked. If the remote server is not active, ACAS will return to the caller node to check whether the method server is registered in its interface repository. If yes, then it will launch (activate and invoke) the server. If this server name is not registered, than it will raise an exception message back to the calling client.

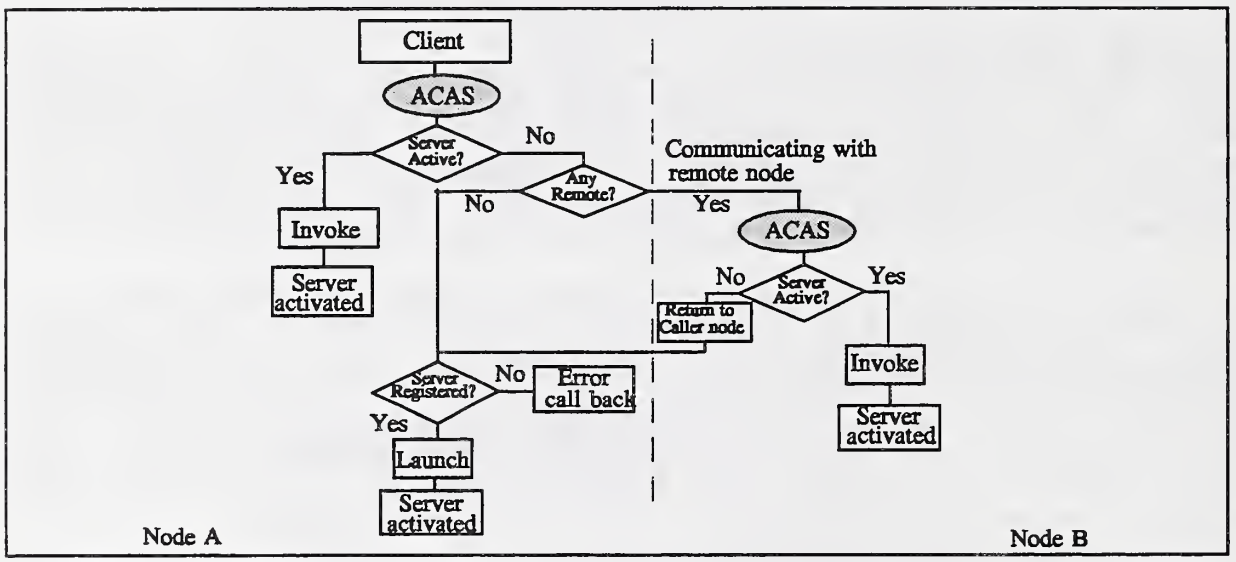

Figure 4.2 Activation Policy of ACAS

4.3.2 Activation Policy of ORBIX

The Activation policy of ORBIX is depicted in Figure 4.3. The client issues a request for a method server. ORBIX first checks whether the needed server is registered in its interface repository. If yes, then it will activate the method server by either invoking, if the method server is in active state, or, if the method server is deactivated, by launching the server for execution. If the needed server is not registered in the local node, it will go to find a remote node where this method server is registered to be invoked and launched. If none is registered, it will raise an exception message back to the calling client.

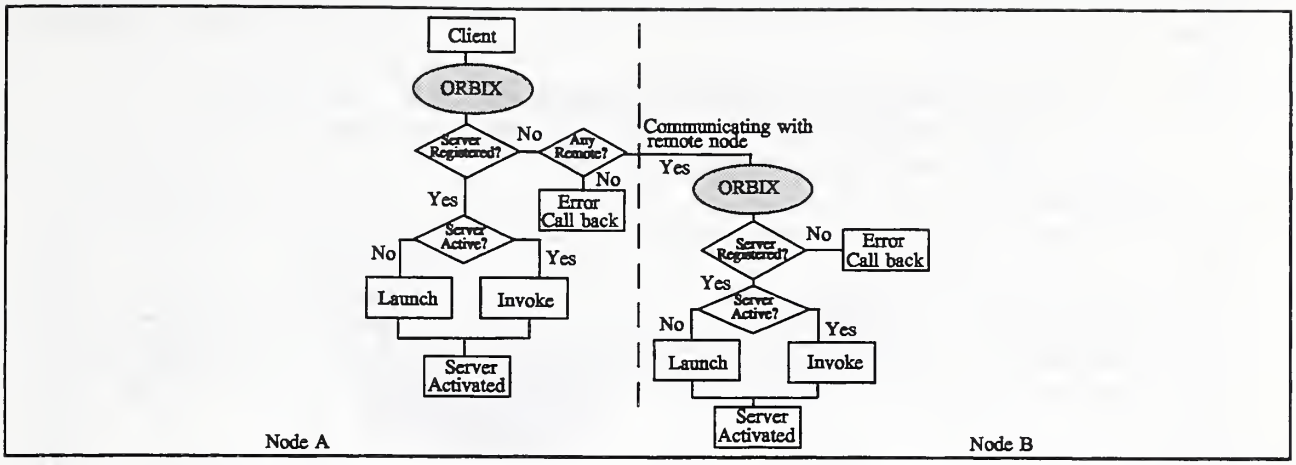

Figure 4.3 Activation Policy of ORBIX

4.4 Test Scenarios

In order to determine the activation policy with different variabilities, several test scenarios were designed and exercised. These tests were defined below and also illustrated with diagrams. The tests were coded and ran individually with the two CORBA products, ORBIX and ACAS. The only exception is in Scenario 4 where both ORB products participated in the same test. The test results from ACAS and ORBIX are discussed in Section 4.5.

Scenario 1: Client, Server and the ORB demon all on the same machine. (See Figure 4.4)

1.1: server registered and active
1.2: server registered but not active
1.3: server not registered

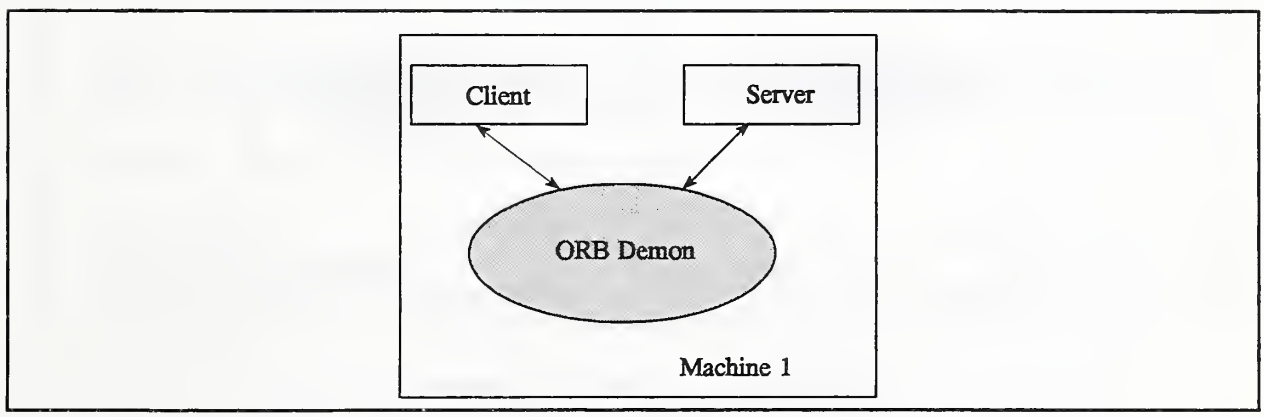

Figure 4.4 Single Machine Resident

Scenario 2: Client on Machine 1, Server and ORB demon on Machine 2. (See Figure 4.5)

2.1: server registered and active
2.2: server registered but not active
2.3: server not registered

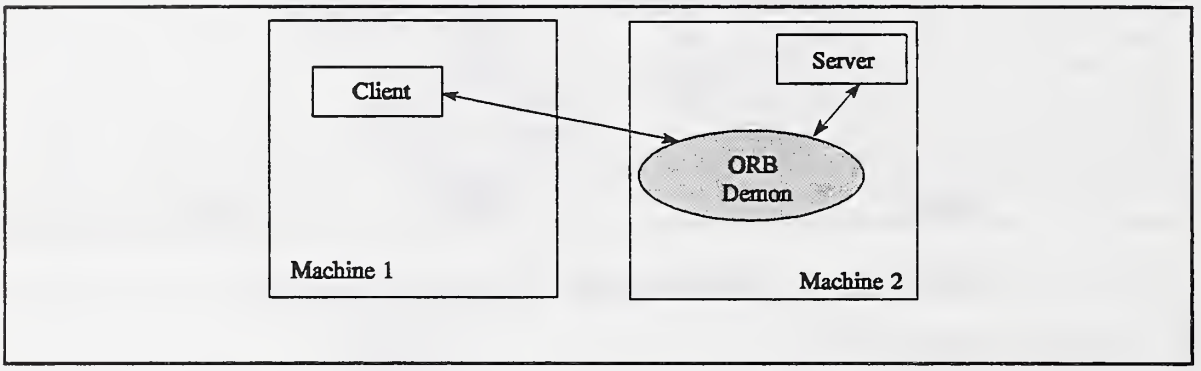

Figure 4.5 Two Machines

Scenario 3: Peer-to-peer ORB connection (same ORB on different machines)

client, server, and demon on Machine 1,
client, server, and demon on Machine 2.
(See Figure 4.6)

3.1: Machine 1 server registered
3.2: Machine 1 server registered but not active
3.3: Machine 1 server not registered, go to machine 2

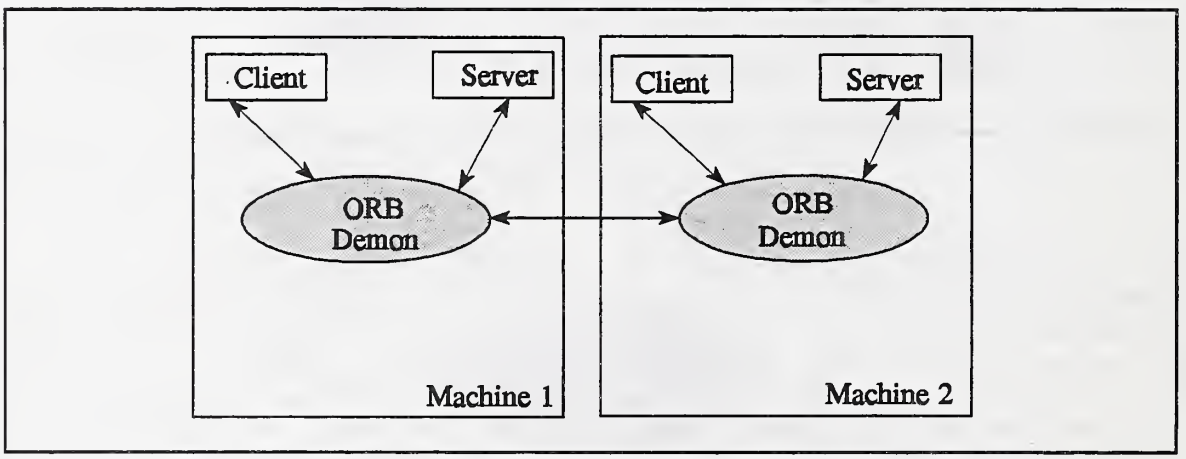

Figure 4.6 Peer to Peer

Scenario 4: Multiple ORB to ORB connections using different ORB products. (See Figure 4.7)

4.1 Server from one ORB becomes the client to another ORB through a gateway.

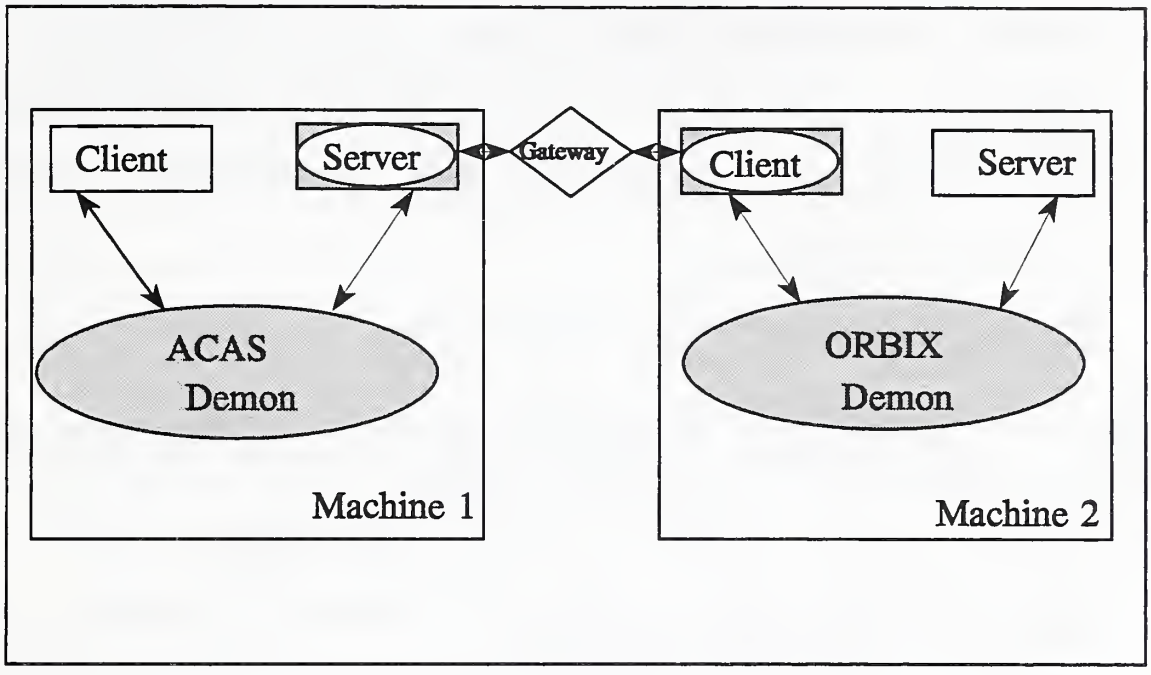

Figure 4.7 Gateway to Foreign ORB

4.5 Test Results for ACAS and ORBIX

Scenario 1: Client, server and ORB programs running on Machine 1.

1.1: server registered and active

ACAS Result: The matched server is invoked and executed.
ORBIX Result: The matched server is invoked and executed.

1.2: server registered but not active

ACAS Result: The matched server is first launched and then executed.
ORBIX Result: The matched server is first launched and then executed.

1.3: server not registered

ACAS Result: The matched server is not found.
ORBIX Result: The matched server is not found.

Scenario 2: Client on Machine 1, Server and demon on Machine 2

2.1: server registered and active

ACAS Result: The registered server is invoked and executed.
ORBIX Result: The registered server is invoked and executed.

2.2: server registered but not active

ACAS Result: The client fails to bind and an error message resulted.
ORBIX Result: The client will find the registered server, which is then launched and executed.

2.3: server not registered

ACAS result: The server is not found.
ORBIX result: The server is not found.

Scenario 3: Peer-to-peer ORB connection with two (but same) ORBs running on two separate machines, either through the same sub-network or remotely to an external network.

3.1: Machine 1 server registered locally and active

ACAS and ORBIX Results: The local server is found and the server is invoked and executed.

3.2: Machine 1 server registered locally but not active

ACAS result: When the local server was found not active, ACAS will go to the remote server to check whether the remote server is active. If found to be active, the remote server is invoked and executed and results returned back to Machine 1. If the remote server is found not active, ACAS will go back to Machine 1's server which will be launched and executed.

ORBIX result: The local server, when found not active, will be automatically launched and executed.

3.3: Machine 1 server not registered (go to Machine 2)

ACAS result: If server not registered locally, ACAS will not go to remote Machine 2, but will report a server not found error message.

ORBIX result: If server not registered locally, ORBIX will go to remote Machine 2 to find out whether the needed server is registered remotely. If yes, remote server will be launched and executed and results will be sent back to Machine 1. If the remote server is also not registered, than an error message will be issued.

Scenario 4: Multiple ORBs communication

4.1 Server from one ORB becomes the client to another ORB's server

Result: ACAS client sends a message to ACAS server, but the ACAS server is the client for ORBIX, who in turn invokes a matched ORBIX method server. This ORBIX method server could be local or remote to another ORBIX demon.

5. CORBA WITH PERSISTENT OBJECT BASE SYSTEMS

This section describes an open, distributed, object-based architecture where plug-and-play of software components can be achieved. Some of the software components which are used for this interoperability experiment are POB systems. The common theme with POB systems is that these systems can be modeled as large-grain, reusable data servers. As data server objects, there is a separation of an object's interface and its implementation. The objects communicate with each other through their interfaces. The interface of an object consists of the functions and data that have been declared for that object. In the case of CORBA, the interface definitions are expressed using IDL statements. The only way to find out something about an object or to ask an object to perform some action is through its interface. On the other hand, an object's implementation is its specific way of carrying out requests to its interface.

5.1 CORBA and POB Architecture

The "open" distributed object-based architecture, as defined for this project, consisted of the following components:

- software which acts as clients (or front-ends) in the form of user interface components,
- a number of persistent database software packages or other form of data sources (or back-ends), in the form of object servers, relational or object DBMSs, legacy systems, and files.
- middle-ware which consists of a CORBA-compliant product that acts as the "glue" for interoperability among different clients and different distributed object servers.

The hardware and software environment for the interoperability demonstration scenarios are presented in Appendix A of this report. The key software components used consisted of two CORBA products: ACAS from DEC and ORBIX from Iona, and three POB systems: Open OODB from Texas Instruments, MATISSE from ADB, and ORACLE from ORACLE.

A set of demonstration script were developed using a simple database application. The sample application, called the "Movie" database, was a subset of the movie data collected by Dr. Gio Wiederhold (formerly of ARPA). The schema of the database is described in Appendix B.

The demonstration scripts can be executed in the World Wide Web using either Mosaic or Netscape. The current demonstration uses the graphical user interface of a Netscape client with Common Gateway Interface (CGI) invoked programs to the various

interoperability test scenarios.[1]

In the investigation of how CORBA can be integrated with POB systems, it was discovered that such integration depends on how "open" the POB modules are, and how POB modules can be "wrapped." For example, CORBA can be considered as an application program of a POB, or a POB can be considered as a method server for CORBA. Some POB systems are designed to be "open" meaning that they expose their internal interfaces so that connecting CORBA directly to these interfaces is possible. On the other hand, some monolithic POB systems are "not open" and thus CORBA can only be connected through their external or public interfaces.

The primary goal of choosing these different test scenarios was to examine what happened when middleware was in various locations in an open distributed object-based architecture. No specific performance type evaluations were conducted for the different test configurations.

5.2 Test Scenarios of CORBA Accessing POB Systems

The interoperability test experiments defined here consists of the following scenarios:

- Scenario 1: CORBA to access Exodus (kernel of Open OODB)
- Scenario 2: CORBA between Exodus client and Exodus server
- Scenario 3: CORBA to access TI OODB
- Scenario 4: CORBA to access MATISSE
- Scenario 5: CORBA to access ORACLE
- Scenario 6: CORBA accessing a distributed processing server which in turn accesses 3 POBs
- Scenario 7: CORBA accessing multiple POBs.

The first three scenarios were designed to show how CORBA can be placed between three different locations by dividing Open OODB into various open modules. Scenarios 4 and 5 were designed to show how to wrap commercially available DBMS with legacy or existing databases. Scenarios 6 and 7 were designed to demonstrate CORBA accessing multiple POB systems. Each of the scenarios will be described and illustrated with diagrams.

The Open OODB developed by Texas Instruments is a prototype POB whose architecture is unique in that it consists of a functional reference model defining a collection of loosely-coupled software components [WELL91]. All of the software components are "open" meaning that the components have public, well-documented

---

[1] In order to exercise the CORBA test demonstration, the server needs to be activated on first use. Therefore, the URL for the demonstration is not published here.

interfaces and thus the components can be changed or replaced to vary functionality or improve performance. The Open OODB can be viewed as a database toolkit consisting of a number of software components that each have an exposed application programming interface and an encapsulated method procedure. For every application, a different set of components can be assembled for plugging and playing.

The kernel of Open OODB uses the storage manager called Exodus which was developed at the University of Wisconsin by Michael Carey and David DeWitt [CARE88]. Exodus was designed to be an extensible database system such that user-added extensions can be supported. The architecture of Exodus, itself, consists of a general-purpose access method storage manager which can be considered as server, and a client program providing query capabilities.

Scenario 1 is illustrated in Figure 5.1. In this scenario the Exodus architecture itself consists of an Exodus client and an Exodus server. Both Exodus client and Exodus server are considered as a single data source. The CORBA is placed on top the Exodus client and the Exodus client's interface is defined in the IDL statements. After the IDL statements are compiled, a client stub program and a server skeleton program are generated. The application code for the CORBA client, which includes the generated client stub, is than written. The application code for the CORBA method server, which includes the generated server skeleton, is also written. Our demonstration script consisted of the client requesting data stored in an Exodus database.

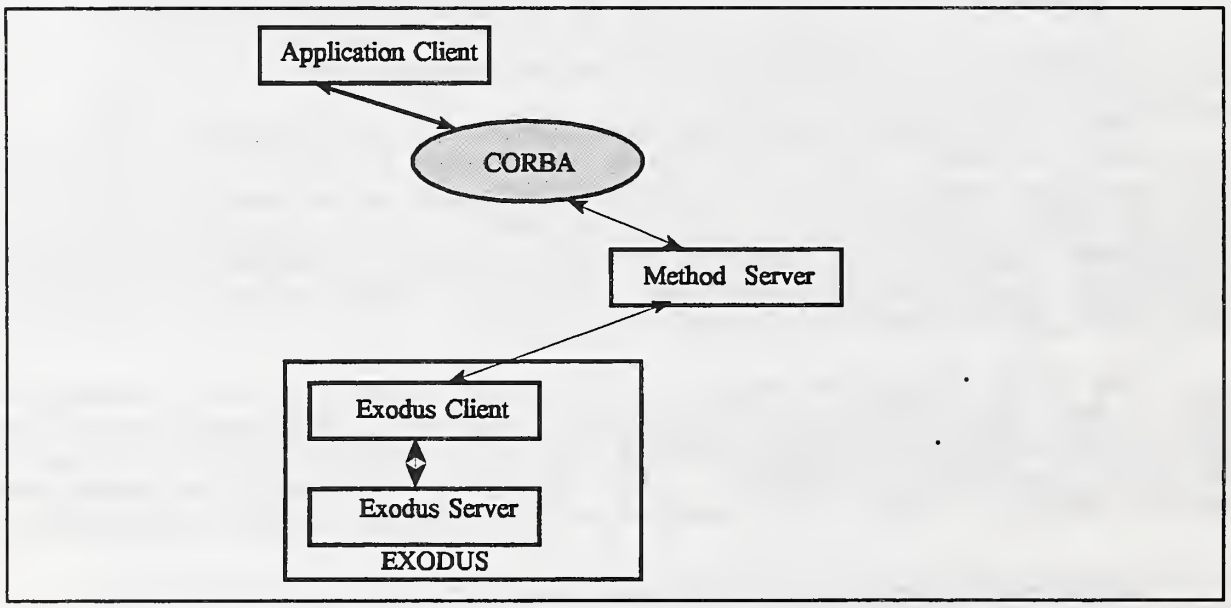

Figure 5.1 CORBA to access Exodus

Scenario 2 is illustrated in Figure 5.2. In this scenario the Exodus database software is split into two separate software components. The Exodus server's interface is defined in the IDL statement to be called by the CORBA's method server. The CORBA's application client program accesses the Exodus client and together they act as CORBA's client program. This scenario is possible because Exodus exposes its server interface to the user.

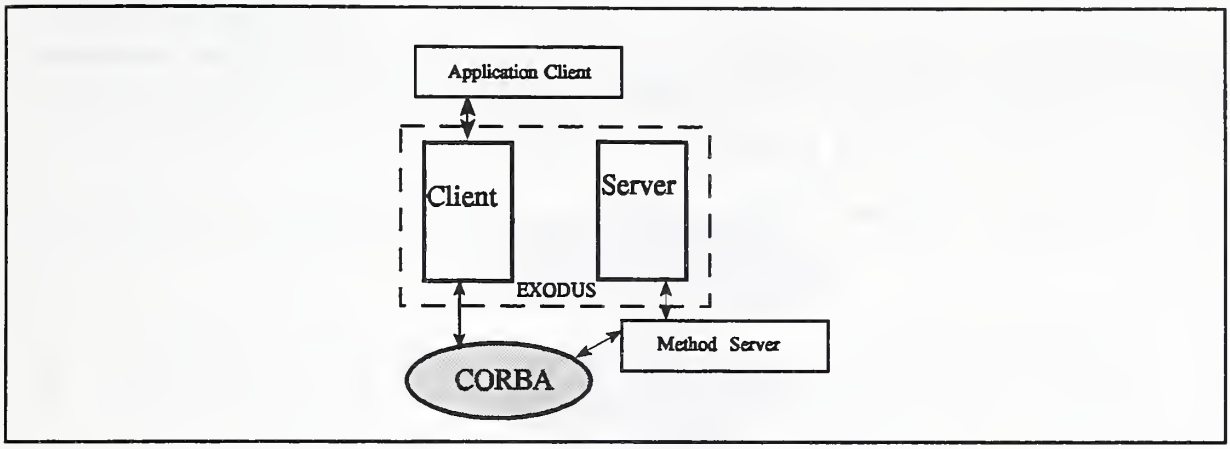

Figure 5.2 CORBA between Exodus client and Exodus server

Scenario 3 is illustrated in Figure 5.3. In this scenario the Open OODB is considered to be a large-grain, single data source, although the Open OODB itself consists of multiple software modules, including the Exodus client and the Exodus server. Our demonstration script consisted of the client requesting data stored in the Open OODB database, and a browse commend requesting multiple data records from the Open OODB database.

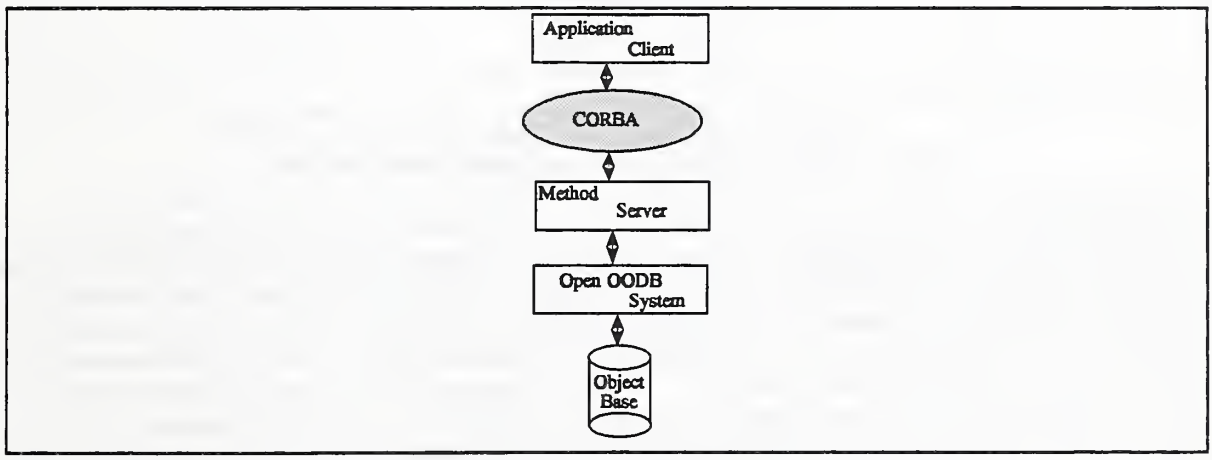

Figure 5.3 CORBA to access open OODB

MATISSE is an object database management system commercially available from ADB, Inc. Although MATISSE's architecture has client and server modules, the only interfaces exposed to the users are the MATISSE commands. Therefore, MATISSE needs to be treated as a large-grain data source to be accessed by the CORBA method server.

Scenario 4 is illustrated in Figure 5.4. In this scenarios the MATISSE database management system and the populated database are considered to be a single data source to be accessed by the CORBA's method server. Our demonstration script consisted of a retrieval operation and a browse operation.

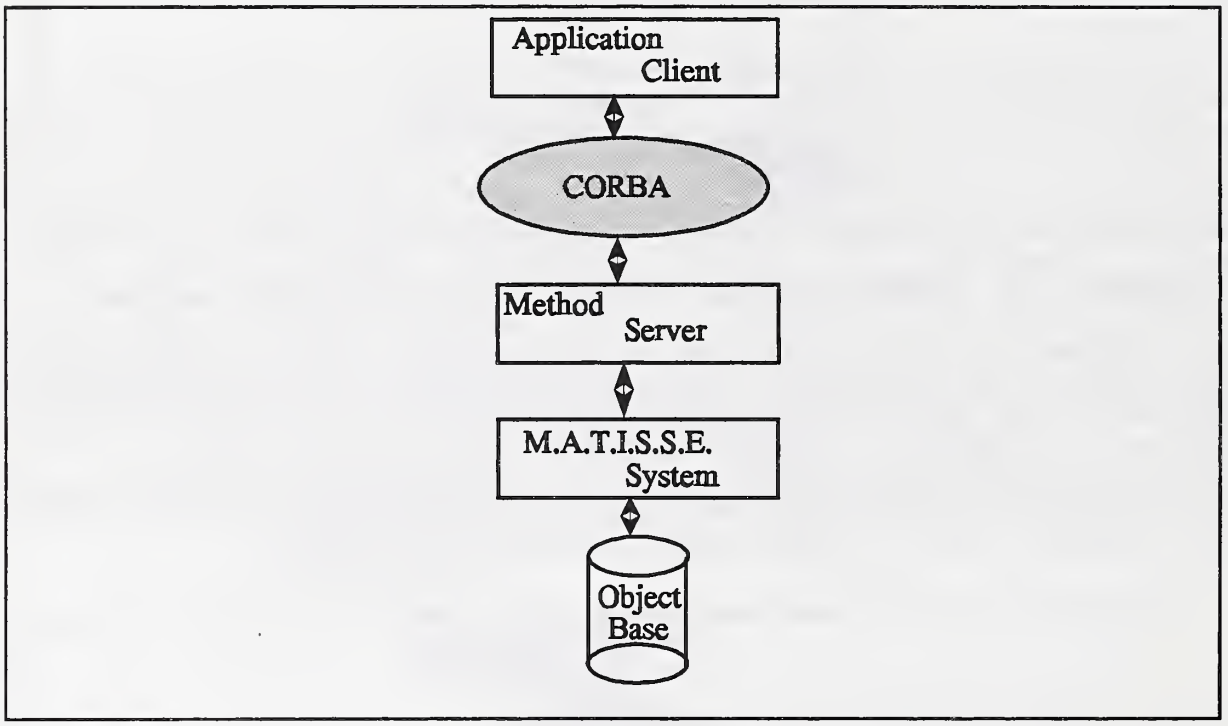

Figure 5.4 CORBA to access MATISSEE

ORACLE is a relational database management system commercially available from the ORACLE Corporation. The ORACLE database and the SQL retrieval application were pre-existing. This test scenario was designed to investigate how legacy software can be "wrapped." Wrapping services act as translators between legacy data sources and other software components. The purpose for developing wrappers is to be able to re-use pre-existing software without having to re-code. In essence, a wrapping service takes a data source and modifies its interface, its data, and/or its behavior so that it is accessible by the outside world, which in our case will be the CORBA method server.

Scenario 5 is illustrated in Figure 5.5. In this scenario, the CORBA's method server program provides the code for "wrapping" the ORACLE DBMS. The wrapping program simply transforms the interface of ORACLE, which in our demonstration script, consisted of SQL retrieval commands to the CORBA's IDL interface statements. In general, the transformation process are translating between different data formats and also between different metadata formats at the machine or file level.

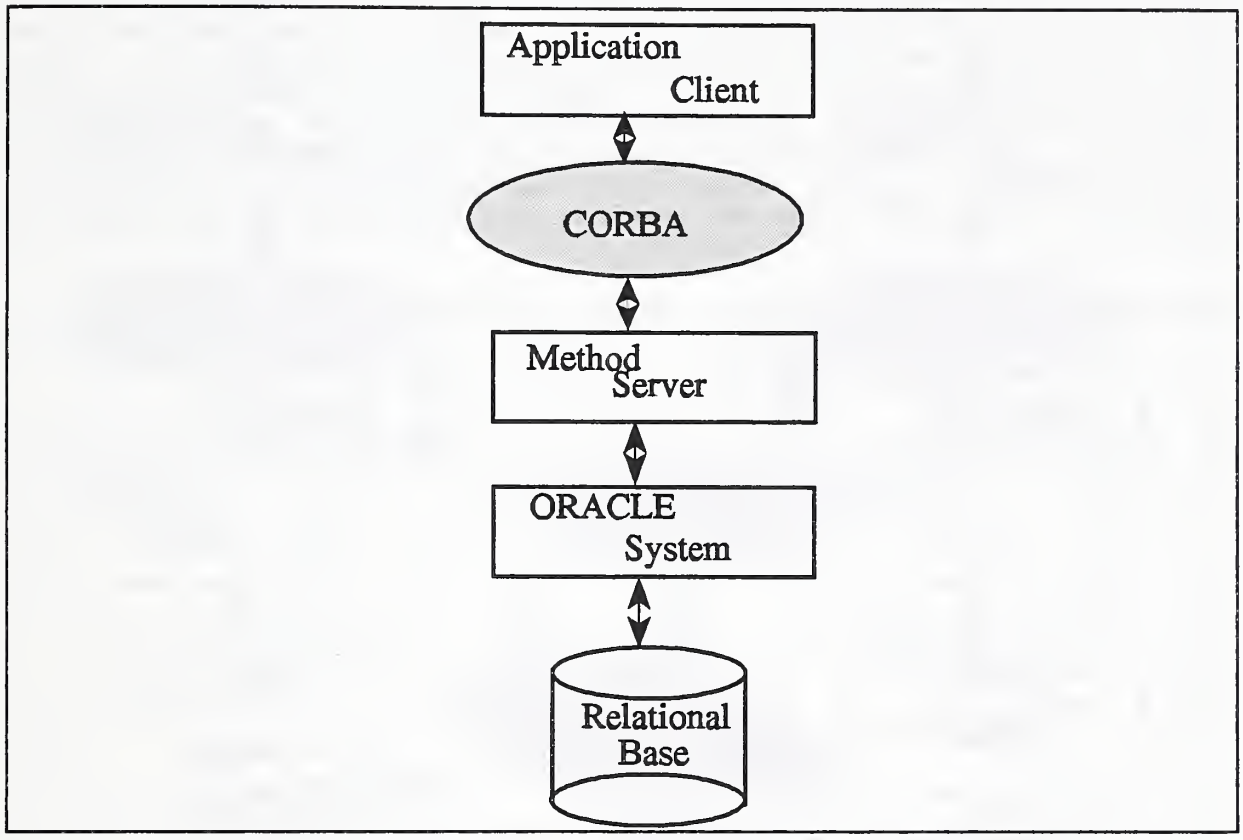

Figure 5.5 CORBA to access ORACLE

To experiment with CORBA interoperability with heterogeneous and multiple data sources, we designed two test scenarios: CORBA accessing a distributed processing server which in turn access the multiple POB systems, and CORBA accessing multiple POB systems individually each having a its own method server. Those multiple POB systems can be on the same machine or can be remotely accessed.

The test database consisted of a uniform movie database schema (see Appendix B), but the data values populated in different POBs were all distinct. This permitted the test user to distinguish where the data record was retrieved from, even if the user did not know the location of the data to be retrieved at the start of the retrieval.

The test architecture consisted of client application programs operating in a graphic user interface environment such as Mosaic or Netscape. The client programs accepted user commands in simple SQL-like query language. The request went to CORBA as the middleware to access method servers which could accommodate wrappers for heterogenous data sources.

Scenario 6 is illustrated in Figure 5.6. In this scenario, all three POB systems: Open OODB, MATISSE, and ORACLE were accessed from the same CORBA's method server. This method server contains software which contains location information so that the query can be analyzed and sent to invoke the appropriate POB systems. These three POB systems could be in the same machine, or they may even be residing on separate machines which could be geographically dispersed. The CORBA itself could be residing in one machine or could be going through peer-to-peer CORBA interconnection in activating the particular CORBA method server such as shown in Figure 5.6.

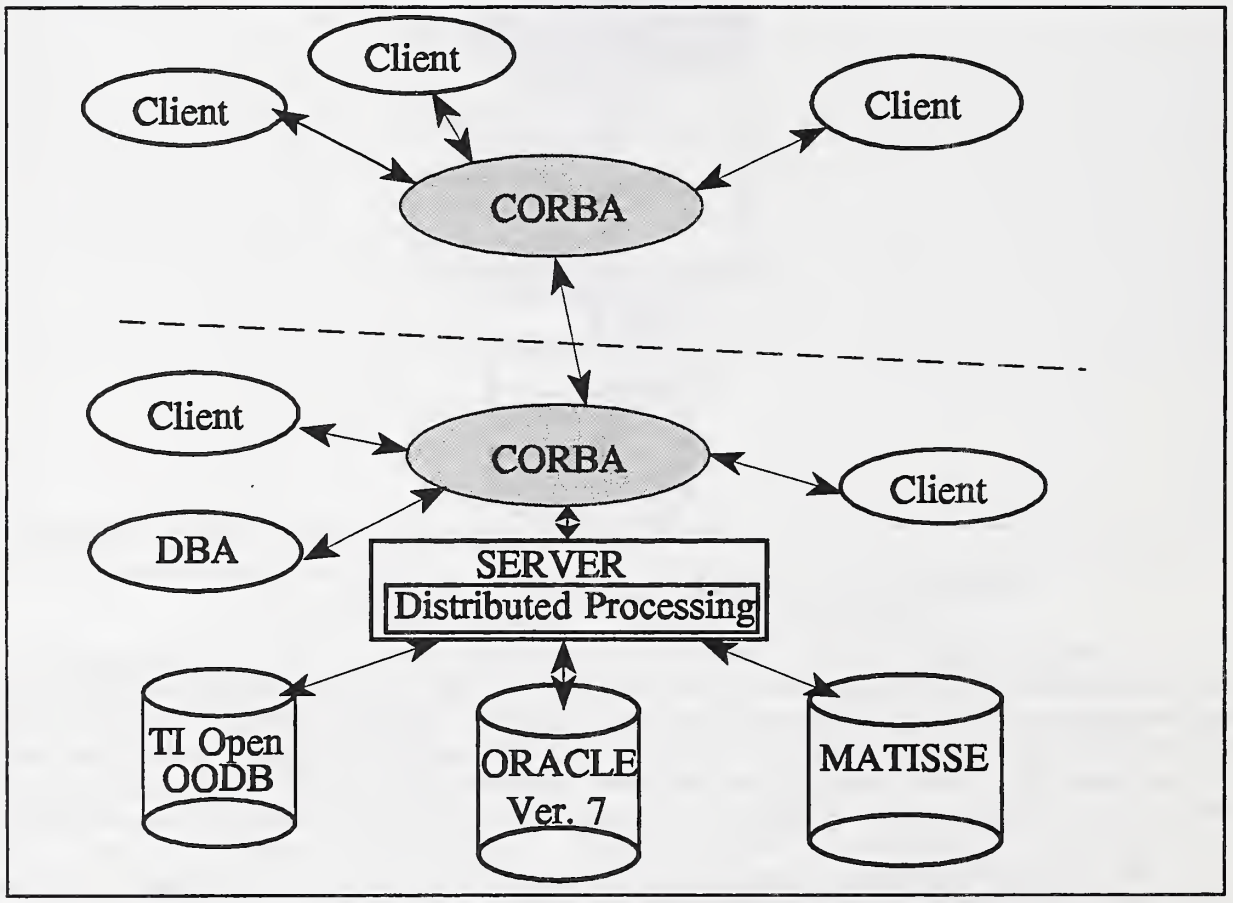

Figure 5.6 CORBA to access multiple POBs

Scenario 7 is illustrated in Figure 5.7. In this scenario, there needs to be multiple CORBA demons, one for each POB. Each POB had its own method server program which functioned as a wrapper to its own database interface system. The user needed to know the location of the data and had to tell the application client program which POB to access for the retrieval or browse of specific data.

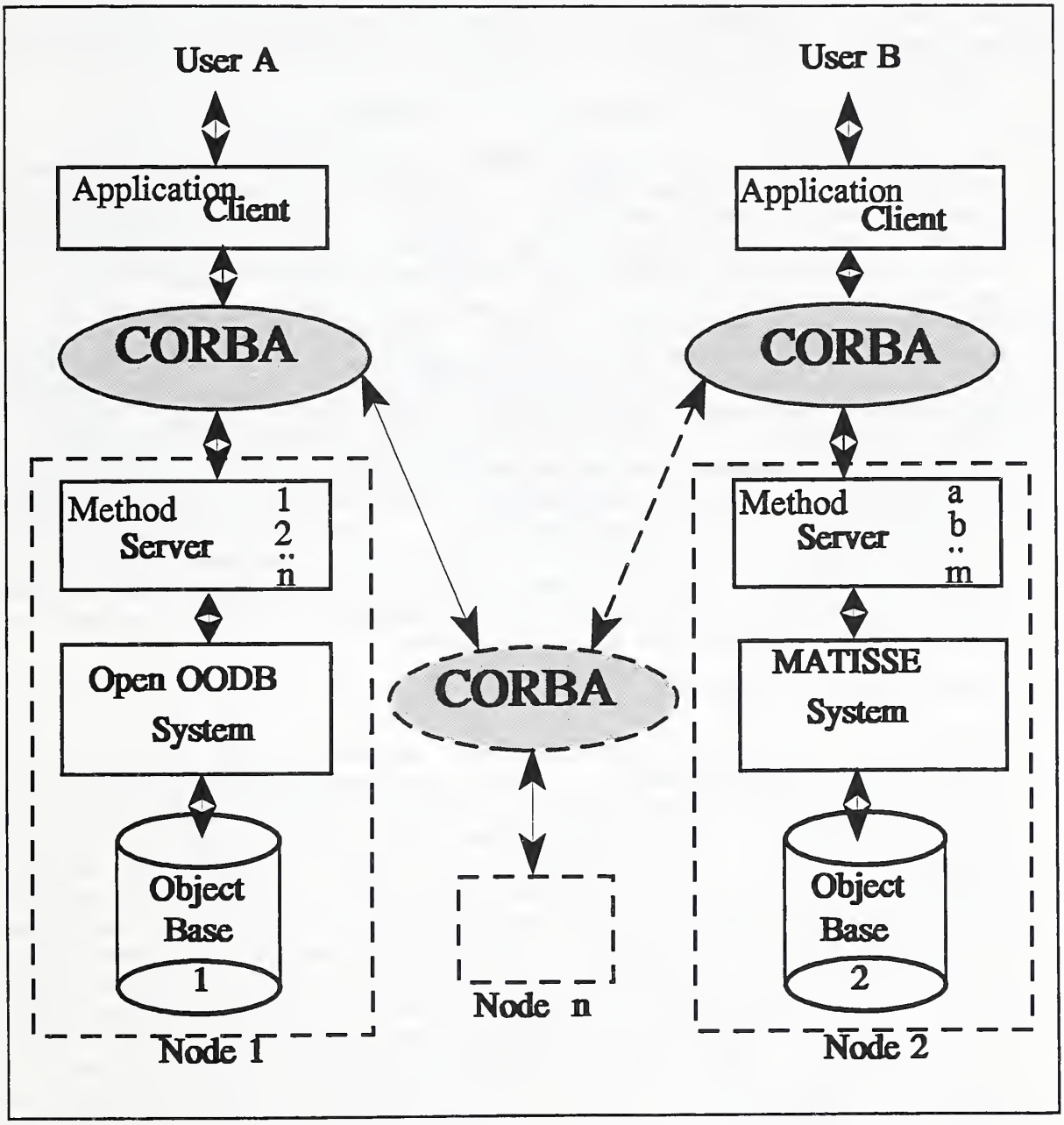

Figure 5.7 CORBA accessing multiple POBs

6. CONCLUSIONS

In this report, we have reviewed the capabilities of object request brokers through actual tests and case studies. We explored a variety of configurations to demonstrate interoperability among distributed, heterogeneous application systems.

The results obtained from these interoperability scenarios reflect the following issues:

- Distributed Object Computing Environment

  Using "objects" and OMG's CORBA technology does provide a tractable way of organizing the complexity of modern computing systems and also serves as a way to simplify distributed processing. In the 1970's, a typical computing installation was a centralized computing resource that housed data and applications accessed by users and applications connected via direct telecommunications links. Today, systems are designed under the client/server paradigm consisting of data resources on servers and client applications residing on local processors that access the server-based data across a local or wide area network. The CORBA technology, sitting between the client and the servers, serves as middleware which permits an application to send parameters and data to, and receive results from, the server process. The job of the object request broker is to manage the interaction between client and server objects. This loose coupling of client and server objects through the use of CORBA supports object reuse and distributed processing architectures.

- Interoperable Objects

  There are two constraints that must be met for an object to be viewed as "interoperable." The first constraint is that the nature of object technology which calls for a separation of the interface to the object from the implementation of the object method(s). The second constraint is that the object provides the code and the data in a form that can be accessed by multiple processors in a heterogeneous environment without regard to physical location. This interoperability mechanism is accomplished through an interface definition language (IDL) which is a declarative language for writing object interfaces only. The IDL expression includes ways of specifying the name of the object, the input parameters and the result types. A mapping between the IDL and whatever programming language is used to implement the client and server objects must be provided. The client implementation language need not be the same as the server implementation language as long as the IDL mapping to both is available. This facilitates the interoperation between client and server applications which

can be across programming language boundaries and ultimately across platform boundaries.

- CORBA Prospects

  Since the CORBA technology is maturing rapidly, the question most frequently asked is "is it a standard?" Within the OMG consortium CORBA is a standard description of an architecture, but it is not a standard for implementation. This does allow implementations to vary in their details. The result is that each implementation of CORBA is a proprietary product. Version 2.0 of the CORBA promises interoperability with different ORB products. CORBA, as demonstrated by these interoperability experiments, can be viewed as a viable technology that enables cross platform, cross programming language, and cross operating system interoperability.

The CORBA architecture appears to provide a useful environment for building distributed application systems. The interoperability experiments performed in this project revealed that, with the use of CORBA, persistent object base systems and legacy systems can be wrapped which allows objects written in any language to be shared and reused. This allows objects based on various languages and platforms to be utilized under the plug-and-play concpet of interoperablility.

REFERENCES

[BOWE95] The Bowen Group Inc., UniSQL's Object-Relational Data Management Technology, Enterprise Reengineering Product Profiles, Available from The Bowen Group Inc. 6168 Evergreen Way, Ferndale, WA 98248-9686.

[CATT94] Cattell R.G.G. (Editor) The Object Database Standards: ODMG - 93, Morgan Kaufmann Publishers, 1994.

[CARE88] Carey, M. J. et al., The EXODUS Extensible DBMS Project: An Overview, Computer Sciences Technical Report #808, University of Wisconsin, Madison, November 1988.

[DABR90] Dabrowski, C., Fong, E., and Yang, D., Object Database Management Systems: Concepts and Features, NIST Special Publication 500-179, April 1990.

[FONG91] Fong, Elizabeth; Sheppard, Charles; and Harvill, Kathryn; Guide to Design, Implementation and Management of Distributed Databases, NIST Special Publication 500-185, February 1991.

[FONG95] Fong, Elizabeth, Persistent Object Base System Testing and Evaluation, NISTIR 5636, April 1995.

[FLEI93] Fleisher, Jeff, "The DISCUS Initiative" Viewgraph presentation at the DISCUS Symposium, August 1994.

[OMG91] Object Management Group, The Common Object Request Broker: Architecture and Specification, OMG Document Number 91.12.1, Revision 1.1, December 1991.

[OMG94] Object Management Group, CORBA products Directory, Available from OMG, Inc. 492 Old Connecticut Path, Framingham, MA 01701, March 1994.

[OODB91] Object-Oriented Database Task Group, Final Report of X3/SPARC/DBSSG/OODBTG, Available from E. Fong, at efong@nist.gov September 1991.

[POST94] PostModern Computing Technologies, Inc., ORBeline User's Guide, available from PostModern Computing Technologies, Inc., 1897 Landings Drive, Mountain View, CA 94043, USA.

[SOLE90] Soley, Richard M. Object Management Architecture Guide 1.0, OMG TC Document 90.9.1, Available from OMG, Inc. 492 Old Connecticut Path, Framingham, MA 01701, Nov. 1990.

[WELL91] Wells, David, DARPA Open Object-Oriented Database Preliminary Architecture Specification, Version 5.0 Texas Instruments Inc. Dallas Texas September 4, 1991.

APPENDIX A

ENVIRONMENT FOR INTEROPERABILITY EXPERIMENTS

The interoperability demonstration scenarios are based on the following environment. Certain commercial software products and vendors are identified in this report for purposes of specific illustration. Such identification does not imply recommendation or endorsement by NIST, nor does it imply that the products identified are necessarily the best ones available for the purpose.

CORBA Products:

- ACA_Services (DEC)
- ORBIX (Iona)

Persistent Object Base Products:

- Open OODB (Texas Instruments)
- MATISSE (ADB)
- ORACLE (ORACLE)

Host Computers and Platforms:

- speckle.ncsl.nist.gov (Sun/Unix)
- Kirk.ncsl.nist.gov (Sun/Unix)
- brew.cme.nist.gov (Sun/Unix)
- titan.cme.nist.gov (Sun/Unix)

User Interface Tools:

- Mosaic
- Netscape
- Xwin/Openwin: command language

Programming Languages:

- C/C++
- Interface Definition Language (CORBA IDL)
- APIs from different products
- Hypertext Markup Language (HTML)

APPENDIX B

THE MOVIE APPLICATION

The sample application schema used for the CORBA demonstration scripts is from the data collected by Dr. Gio Wiederhold called the "Movie" database (See diagram).